Introducing Zend Framework 3

Rob Allen ~ @akrabat ~ February 2016

What did ZF2 give us?

- Dependency injection
- Event-driving architecture
- Standalone, first-class modules

What's wrong with ZF2?

The PHP world has changed since 2012

So what's the ZF3 story?

The ZF3 story

- Componentisation
- Performance and usability
- MVC improvements!
- Focus on PSR-7, Interoperability & Middleware

PHP 5.5

Components

Components

- Separate repositories

Components

- Separate repositories
- PSR-4 structure for source and tests

Components

- Separate repositories
- PSR-4 structure for source and tests
- Separate evolution

Components

- Separate repositories
- PSR-4 structure for source and tests
- Separate evolution
- Documentation in repository

Components

- Separate repositories
- PSR-4 structure for source and tests
- Separate evolution
- Documentation in repository
- All issues in the right place on GitHub

Components

- Separate repositories
- PSR-4 structure for source and tests
- Separate evolution
- Documentation in repository
- All issues in the right place on GitHub
- More maintainers

ZF MVC framework

MVC improvements

- ZF2 is now a meta package

The framework will selectively upgrade, but each component can evolve separately Easier to slim down to just the components needed Leads to Use-case specific skeletons

MVC improvements

- ZF2 is now a meta package
- ZF3 will have fewer dependencies - just what's needed for MVC

MVC improvements

- ZF2 is now a meta package
- ZF3 will have fewer dependencies - just what's needed for MVC
- Managed BC breaks

MVC improvements

- ZF2 is now a meta package
- ZF3 will have fewer dependencies - just what's needed for MVC
- Managed BC breaks
- First 3.0 MVC components:
    - `ServiceManager`
    - `EventManager`

Other components : ZendHydrator and ZendCode are at 3.0 (Code supports PHP 5.5, 5/6 & 7 (scalar typehints, return typehints, generators, and variadics.)

Zend\ServiceManager 3.0

- Container-interop

Zend\ServiceManager 3.0

- Container-interop
- Consistent interfaces

Zend\ServiceManager 3.0

- Container-interop
- Consistent interfaces
- Re-use factories for multiple named services

Zend\ServiceManager 3.0

- Container-interop
- Consistent interfaces
- Re-use factories for multiple named services
- New method: `build()` for factories

Zend\ServiceManager 3.0

- Container-interop
- Consistent interfaces
- Re-use factoriers for multiple named services
- New method: `build()` for factories
- Immutable

Zend\ServiceManager 3.0

- Container-interop
- Consistent interfaces
- Re-use factoriers for multiple named services
- New method: `build()` for factories
- Immutable
- Fast! (4x to 20x faster!)

Zend\ServiceManager 3.0

- Container-interop
- Consistent interfaces
- Re-use factoriers for multiple named services
- New method: `build()` for factories
- Immutable
- Fast! (4x to 20x faster!)
- Mostly backwards compatible

Zend\ServiceManager 3.0 Key Changes

- Service name are case sensitive and no longer normalised
- Constructor now takes an array, not a `Config` object
- New interfaces for factories: `__invoke()`
- PluginManager factories are now passed the parent ServiceManager

Zend\EventManager 3.0

- Fast! (4x to 15x faster!)
- Usability improvements to `trigger()`
- Mostly backwards compatible still

Zend\EventManager 3.0 Key Changes

- `GlobalEventManager` and `StaticEventManager` have been removed

Zend\EventManager 3.0 Key Changes

- `GlobalEventManager` and `StaticEventManager` have been removed
- Listener aggregates have been removed

Zend\EventManager 3.0 Key Changes

- `GlobalEventManager` and `StaticEventManager` have been removed
- Listener aggregates have been removed
- `EventManager::__construct()` signature has changed

Zend\EventManager 3.0 Key Changes

- `GlobalEventManager` and `StaticEventManager` have been removed
- Listener aggregates have been removed
- `EventManager::__construct()` signature has changed
- `trigger()` changes:

```
trigger($eventName, $target = null, $argv = [])
triggerUntil(callable $callback, $eventName, $target = null, $argv = [])
triggerEvent(EventInterface $event)
triggerEventUntil(callable $callback, EventInterface $event)
```

Zend\Mvc 3.0

- Updated for zend-servicemanger 3.0 changes
- Updated for zend-eventmanger 3.0 changes
- New `MiddlewareListener` and PSR-7 bridge

It's bascially the *same*!

Where is the PHP community going?

The future

- Dependence on abstractions: PSR-7, PSR-3, container-interop, etc
- Building applications from components in Packagist
- The framework should get out of the way of your code

PSR-7, Interoperability & Middleware

It's all about HTTP

Request:

```
{METHOD} {URI} HTTP/1.1
Header: value1,value2
Another-Header: value

Message body
```

Response:

```
HTTP/1.1 {STATUS_CODE} {REASON_PHRASE}
Header: value

Message body
```

Current PHP

Request:

- $_SERVER, $_GET, $_POST, $_COOKIE, $_FILES
- `apache_request_headers()`
- `php://input`

Response:

- `header()`
- `echo` (& `ob_*()` family)

PSR-7

It's just some interfaces

- `RequestInterface` (& `ServerRequestInterface`)
- `ResponseInterface`
- `UriInterface`
- `UploadedFileInterface`

Two key things about PSR-7

Key feature 1: Immutability

`Request`, `Response`, `Uri` & `UploadFile` are *immutable*

```
$uri = new Uri('https://api.joind.in/v2.1/events');
$uri2 = $uri->withQuery('?filter=upcoming');

$request = (new Request())
    ->withMethod('GET')
    ->withUri($uri2)
    ->withHeader('Accept', 'application/json')
    ->withHeader('Authorization', 'Bearer 0873418d');
```

Key feature 2: Streams

Message bodies are *streams*

```
$body = new Stream();
$body->write('<p>Hello');
$body->write('World</p>');

$response = (new Response())
    ->withStatus(200, 'OK')
    ->withHeader('Content-Type', 'application/header')
    ->withBody($body);
```

Diactoros

ZF's PSR-7 implementation

Diactoros

- Complete PSR-7 implementation

Diactoros

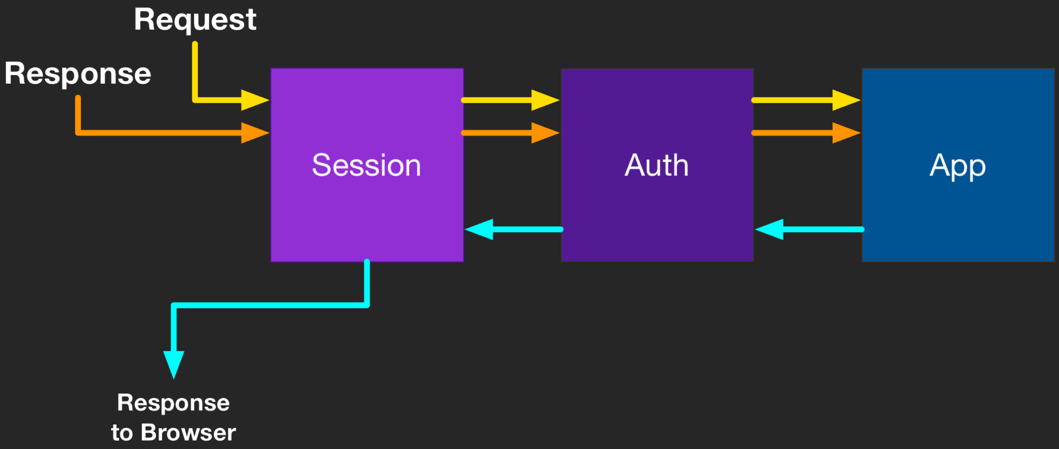
- Complete PSR-7 implementation
- Specialised Responses: JSON, Empty & Redirect

Diactoros

- Complete PSR-7 implementation
- Specialised Responses: JSON, Empty & Redirect
- Bridges:
  - Used by Symfony for their PSR-7 bridge
  - zend-psr7bridge: ZF3's PSR-7 to zend-http bridge

Middleware

Middleware



Middleware

```
function (ServerRequestInterface $request, ResponseInterface $response,
    callable $next = null) : ResponseInterface
{
    // do something before

    // call through to next middleware
    if ($next) {
        $response = $next($request, $response);
    }

    // do something with $response after

    return $response;
}
```

Writing middleware

Pattern:

- Optionally modify the received request and response
- Optionally invoke the next middleware
    - Optionally modify the returned response
- Return the response to the previous middleware.

Stratigility

ZF's Middleware implementation

Stratigility

- Dispatches a stack of middleware

Stratigility

- Dispatches a stack of middleware
- Middleware format:
    - Any `callable`
    - `Zend\Stratigility\MiddlewareInterface`

```
public function __invoke(
    ServerRequestInterface $request,
    ResponseInterface $response,
    callable $out = null
) : ResponseInterface;
```

ErrorMiddleware

Pass error as third parameter to `$next`:

```
return $next($request, $response, $error);
```

ErrorMiddleware

Pass error as third parameter to `$next`:

```
return $next($request, $response, $error);
```

Handle like this:

```
function ($error,
    ServerRequestInterface $request,
    ResponseInterface $response,
    callable $out
);
```

or `Zend\Stratigility\ErrorMiddlewareInterface`

Path segregation:

```
use Zend\Stratigility\MiddlewarePipe();
$app = new MiddlewarePipe();
$app->pipe($mw1);                    // always evaluate
$app->pipe('/blog', $blogMw);        // only if path matches
$app->pipe('/contact', $contactMw);
$app->pipe($outputMw);

$server = Server::createServer($app, …);
$server->listen();
```

Nesting Middleware

Compose middleware together based on path:

```
$blog = new MiddlewarePipe();
$blog->pipe('/post', $postMw);
$blog->pipe('/feed', $rssMw);
$blog->pipe('/', $listMw);

$app = new MiddlewarePipe();
$app->pipe('/blog', $blog);
```

Middleware wrappers

```
$app->pipe('/', $homepage);                     // Static HTML
$app->pipe('/customer', $zf2Middleware); // ZF2
$app->pipe('/products', $zf1Middleware); // ZF1
$app->pipe('/api', $apigility);               // Apigility
$app->pipe('/user', $userMiddleware);    // 3rd party
```

What about routing?

(& DI container, etc…)

Integration with ZF-MVC

Routing to Middleware via the new `MiddlwareListener`:

```
'oauth' => [
    'type' => 'Literal',
    'options' => [
        'route' => '/oauth',
        'defaults' => [
            'middleware' => OauthMiddleware::class,
        ],
    ],
],
```

Expressive

ZF's micro framework

Expressive

- Provides and consumes a routing interface
- Pulls matched middleware from `ContainerInterface`
- Provides an optional templating interface
- Provides error handling

Agnostic

Router:

- FastRoute, Aura.Router or Zend Router

DI Container:

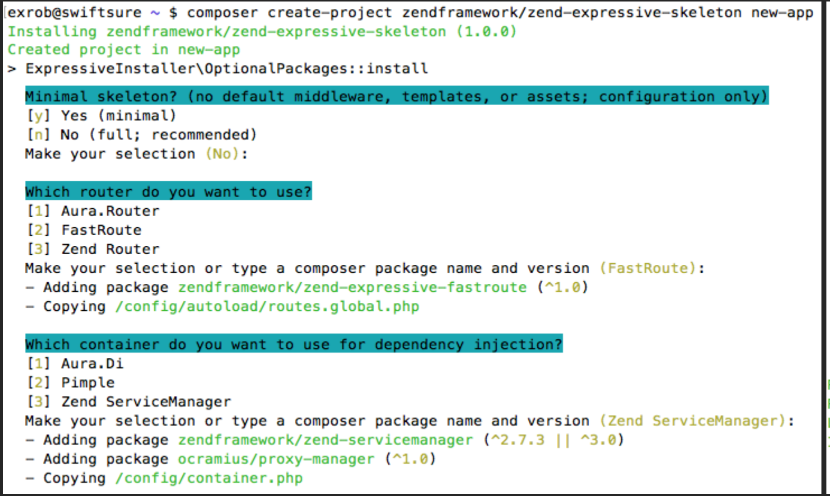
- Zend ServiceManager, Pimple, Aura.Di (or any container-interop DIC)

Template:

- Plates, Twig or Zend View

Installation

```
$ composer create-project zendframework/zend-expressive-skeleton new-app
```

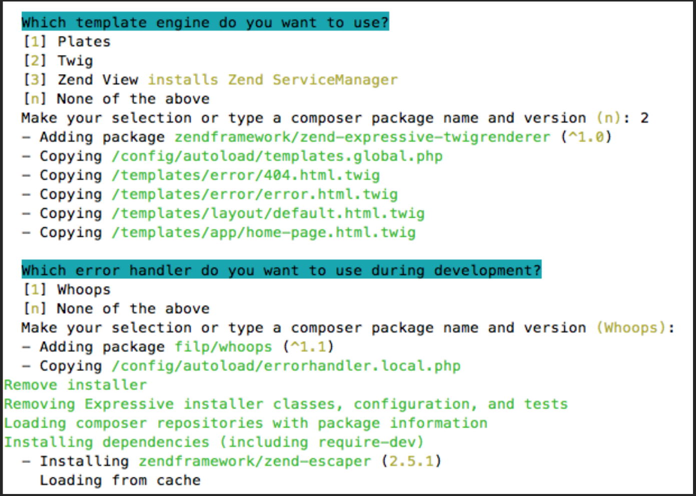


```
exrob@swiftsure ~ $ composer create-project zendframework/zend-expressive-skeleton new-app
Installing zendframework/zend-expressive-skeleton (1.0.0)
Created project in new-app
> ExpressiveInstaller\OptionalPackages::install

  Minimal skeleton? (no default middleware, templates, or assets; configuration only)
  [y] Yes (minimal)
  [n] No (full; recommended)
  Make your selection (No):

  Which router do you want to use?
  [1] Aura.Router
  [2] FastRoute
  [3] Zend Router
  Make your selection or type a composer package name and version (FastRoute):
  - Adding package zendframework/zend-expressive-fastroute (^1.0)
  - Copying /config/autoload/routes.global.php

  Which container do you want to use for dependency injection?
  [1] Aura.Di
  [2] Pimple
  [3] Zend ServiceManager
  Make your selection or type a composer package name and version (Zend ServiceManager):
  - Adding package zendframework/zend-servicemanager (^2.7.3 || ^3.0)
  - Adding package ocramius/proxy-manager (^1.0)
  - Copying /config/container.php
```



```
  Which template engine do you want to use?
  [1] Plates
  [2] Twig
  [3] Zend View installs Zend ServiceManager
  [n] None of the above
  Make your selection or type a composer package name and version (n): 2
  - Adding package zendframework/zend-expressive-twigrenderer (^1.0)
  - Copying /config/autoload/templates.global.php
  - Copying /templates/error/404.html.twig
  - Copying /templates/error/error.html.twig
  - Copying /templates/layout/default.html.twig
  - Copying /templates/app/home-page.html.twig

  Which error handler do you want to use during development?
  [1] Whoops
  [n] None of the above
  Make your selection or type a composer package name and version (Whoops):
  - Adding package filp/whoops (^1.1)
  - Copying /config/autoload/errorhandler.local.php
Remove installer
Removing Expressive installer classes, configuration, and tests
Loading composer repositories with package information
Installing dependencies (including require-dev)
  - Installing zendframework/zend-escaper (2.5.1)
    Loading from cache
```

Hello world

```php
use Zend\Expressive\AppFactory;

$app = AppFactory::create();

$app->get(
  '/hello/{name}',
  function ($request, $response, $next) {
    $name = htmlentities($request->getAttribute('name'));
    $response->getBody()->write("<p>Hello, $name!</p>");
    return $next($request, $response);
  }
);

$app->pipeRoutingMiddleware();
$app->pipeDispatchMiddleware();
$app->run();
```

Middleware pipes

```
$app->get('/', $homepageMiddleware);
$app->get('/contact', $contactMiddleware);

$app->pipe($sessionMiddleware);
$app->pipe($authMiddleware);
$app->pipeRoutingMiddleware();
$app->pipeDispatchMiddleware();

$app->run();
```

Named routes

3rd parameter:

```
$app->get('/books/{id}', $getBookAction, 'book');
```

Build URI:

```
$url = $router->generateUri('edit', ['id' => 1]);
```

Views

No templating by default. Abstracted via `Zend\Expressive\Template\TemplateRendererInterface`

```
$html = $templates->render('book::detail', [
    'layout' => 'master',
    'book' => $bookEntity,
]);

return new HtmlResponse($html);
```

Why Expressive?

- Performance

Why Expressive?

- Performance
- Developer experience

Why Expressive?

- Performance
- Developer experience
- Reusable middleware

This is the ZF3 era

The ZF3 era

- Separate components
- ZF2 MVC with performance improvements
- Stratigility PSR-7 middleware foundation
- Expressive micro framework

Questions?

https://joind.in/talk/6f3ba

Thank you!

https://joind.in/talk/6f3ba

Rob Allen - http://akrabat.com - @akrabat